

REMARKS

Claims 1-40 were pending prior to entry of this response. The oath or declaration was objected to under MPEP 602.01 and 602.02. The abstract was objected to under MPEP 608.01(b). The disclosure was objected to for informalities. Claim 22 was objected to for informalities. Claims 9-10, 21, and 22 were rejected under 35 U.S.C. 112. Claims 1-16, 18-20, and 27-40 were rejected under 35 U.S.C. 102(b). Claims 17 and 21-26 were rejected under 35 U.S.C. 103(a).

The abstract has been amended to correct some minor informalities. Claims 27, 34-35 and 37-40 have been cancelled. To facilitate prosecution, Claims 1, 3, 7, 9, 15, 23, 28-31 and 36 have been amended to correct informalities and to further clarify the claimed invention. Claims 41-47 have been added to more particularly claim the subject matter of the present invention. No new matter has been added to the application. Applicants respectfully request reconsideration of the objections and rejections in view of the following remarks.

A. Objection to the Declaration

The office action alleged that the oath or declaration was defective under MPEP 602.01 and 602.02. The office action stated that the oath or declaration was defective because domestic priority under 35 U.S.C. 119(e) to provisional application 60/116,437 had not been claimed. Applicants have not made a claim of priority to any provisional application and respectfully submit that the objection to the declaration be withdrawn.

B. Objection to the Abstract

The office action objected to the abstract under MPEP 608.01 for having more than 150 words and containing an incomplete sentence. The abstract has been amended to correct these informalities. Applicants respectfully request that the objections be withdrawn.

C. Objection to the Specification

The office action objected to the specification because "there is no section labeled 'Cross-Reference to Related Applications' claiming domestic priority under 35 U.S.C. §119(e) (to a provisional application, 60/116,437)." Office Action, page 2. As stated above, applicants have not made a claim of priority to any provisional application and respectfully request that the objection to the specification be withdrawn.

D. Rejections under 35 U.S.C. 112

Claims 9-10, 21 and 22 were objected to due to informalities, which the applicants determined to be typographical errors. Claims 9 and 21 have been amended to correct these typographical errors. Applicants respectfully request that the rejections to the claims be withdrawn.

E. Claims Rejected under 35 U.S.C. 102

Claims 1-16, 18-20, and 27-40 were rejected under 35 U.S.C. 102(b) as being anticipated by Fortin, USPN 5,528,753. Fortin describes a system and method for monitoring a target routine in a program executable on a computer system where the program is stripped of any linkable information. Specifically, Fortin describes the use of an entry routine and an exit routine to monitor a particular target routine in a stripped program. (Fortin, Figures 2 and 3). Referring to Figure 3 in Fortin, when target routine 204 is called, control is passed to target

routine 204. Then, the control is temporarily passed to entry routine 210, which collects information desired by a monitor and returns the control to target routine 204. When the control is returned, target routine 204 executes and exits. The control is temporarily passed to exit routine 212 before exiting from target routine 204 so that exit routine 212 can collect additional data. (Fortin, column 4, lines 41-53). Thus, according to Fortin, control must first pass to the target routine and must then pass to the monitoring routines. Also, the target routine always executes in conjunction with the monitoring routines.

Unlike Fortin, the present invention does not require initially passing control to an original function before passing the control to a user-supplied function. Rather, the present invention modifies an executable file that includes an original function with a user-supplied function. As described in the specification, the executable file is modified to replace, substitute or instrument the access to the original function with an access to the user-supplied function. The invention also retains information about the original function so that the user-supplied function may call the original function if desired. (Applicants' Application, page 11, lines 7-12). However, in contrast with Fortin, the present invention does not require the original function to execute or passing control to monitoring routines.

Claims 1-14

The Office Action rejected independent Claim 1 by citing a procedure described in Fortin for monitoring a routine. As amended, Claim 1 is directed to a computerized method for creating an instrumented executable file. The computerized method includes modifying an executable file having an original function with a user-supplied function; and retaining access information of the original function, the access information enabling the user-supplied function

to invoke the original function. The method recited in Claim 1 is significantly different from the procedure in Fortin cited by the Office Action.

The procedure in Fortin cited by the Office Action monitors a target routine by passing control to other routines within the target routine. However, unlike the method recited in Claim 1 of the applicants' application, the procedure in Fortin does not include modifying an executable file having an original function with a user-supplied function. Instead, the procedure described in Fortin simply passes control of a program to some monitoring routines and the target routine always executes. The procedure in Fortin also fails to describe retaining information about the target routine subsequent to replacing the routine. Thus, the Fortin procedure is different and, more significantly, teaches away from the method in Claim 1.

Moreover, the method recited in Claim 1 can be applied to executable file that includes functions in a dynamic link library, unlike the procedure in Fortin. Throughout the specification, the present invention is described in detail as to how it can be applied to an executable file having an original function and a user-supplied function that are linked to it. For example, the original function and the user-supplied function may be placed in a dynamic link library. (See applicant's application, page 4 lines 4-14 and page 14, lines 1-3). Dependent Claim 3 particularly claims this aspect of the invention and recites that "the user-supplied function is in a dynamic link library." In contrast, Fortin particularly disclaims linked objects. The invention in Fortin is only intended for software programs that are stripped of any linkable information. (Fortin, column 3, lines 1-2). In particular, because stripped objects do not have a symbol table, the instrumentation library described in Fortin cannot simply be linked to the objects. (Fortin, column 5, lines 27-29). Thus, Fortin also fails to describe the method recited in Claims 1 and 3, which works with linked functions.

For the reasons stated above, applicants respectfully submit that the invention recited in independent Claim 1 and dependent Claim 3 are not anticipated or rendered obvious by Fortin and are allowable. Dependent Claims 2 and 4-14 are also allowable for at least the same reasons.

Claims 15-20

The Office Action rejected independent Claim 15 by essentially citing the same procedure in Fortin that the Office Action used to reject Claim 1. Independent Claim 15, as amended, is directed to a computerized method for executing an instrumented executable file. The method includes modifying the instrumented executable file having an original function with a user-supplied function. The user-supplied function has a jump to the original function. The method also includes saving the address of the original function in a threaded local storage variable and invoking the user-supplied function using the address. None of these steps is described in Fortin.

As mentioned above in the discussion regarding Claim 1, Fortin merely describes a procedure for monitoring a target routine by passing control to other routines within the target routine. Fortin does not disclose modifying an executable file with a user-supplied function where, as described in the specification, the executable file is modified to replace, substitute or instrument the access to the original function with an access to the user-supplied function. Furthermore, Fortin fails to mention the use of threaded local storage (TLS) variable and saving the address of the original function as a TLS. Thus, applicants respectfully submit that the invention recited in independent Claim 15 is not anticipated or rendered obvious by Fortin and is allowable. Dependent Claims 16-20 are also allowable for at least the same reasons.

The Office Action also rejected Claim 16, which includes creating a master lookup table at initialization wherein the master lookup table associates the base address of the instrumented executable file to the address of a function lookup table in the instrumented executable file. As described in the specification, the master lookup table is used to locate the function lookup table and the function lookup table is used to match the name of the original function to the function's address. (Applicants' application, page 18). Neither of the two tables is described by Fortin.

The section cited by the Office Action to reject Claim 16 merely describes a demultiplexor entry used for monitoring a target routine. In particular, the demultiplexor entry serves to direct an instrumentation call to the appropriate common and user specific entry and exit routines. (Fortin, column 5, lines 49-51). However, the demultiplexor entry is not a function table recited in Claim 16 that matches the name of an original function to the function's address. Thus, applicants respectfully submit that dependent Claim 16 is allowable for at least this additional reason.

The Office Action also rejected Claim 19, which includes invoking the original function from within the user-supplied function using the threaded local storage variable. As mention above, Fortin fails to disclose the use of threaded local storage variables. Also, Fortin merely describes monitoring a target routine in a stripped object by passing control to monitoring routines while the target routine is executing. Fortin does not describe invoking an original function from within a user-supplied function, as recited in Claim 19. Applicants respectfully submit that dependent Claim 19 is allowable for at least this additional reason.

Claims 28-29 and 36

The Office Action rejected Claims 28-29 and 36 using the same basis as those used for rejecting Claim 1. Accordingly, applicants respectfully submit that Claims 28-29 and 36, as amended, are allowable for at least the same reasons stated above regarding Claim 1.

Claim 30

Claim 30, as amended, is directed to a computerized system that includes

an executable file having a jump to an original function, the original function having an identity comprising a name and a parameter prototype;

a first software component having a user-supplied function that includes a jump to the original function; and

a second software component for:

receiving the identity of the original function;

receiving the identity of the user-supplied function;

instrumenting the executable file by modifying the identity of the original function with the identity of the user-supplied function; and

storing the original function address in the executable file in association with the name of the original instrumented function.

As discussed above, Fortin merely describes a system and method for monitoring a target routine in a stripped object by passing control within the target routine to some monitoring routines. Fortin does not describe a component for "instrumenting the executable file by modifying the identity of the original function with the identity of the user-supplied function" or "storing the original function address in the executable file in association with the name of the original instrumented function." Thus, applicants respectfully submit that Claim 30 is not anticipated by Fortin and is allowable.

Claim 31

Claim 31, as amended, is directed to a computerized system that includes:

a first module of machine-readable code comprising:

a call to an original function, the call being directed to a user-supplied function; and

a first data structure associating the identity of the original function with the location of the original function; and

a second module comprising the user-supplied function, linked to the first module and a jump to the original function.

Thus, the second module is linked to the first module.

As discussed above, Fortin only discloses a system and method for monitoring a stripped object. Nothing in Fortin describes a system that includes a first module that is linked to a second module having a user-supplied function, as recited in Claim 31. Thus, applicants respectfully submit that Claim 31 is not anticipated by Fortin and is allowable. Dependent Claims 32-33 are allowable for at least the same reasons.

F. Claims Rejected under 35 U.S.C. 103Claim 17

Claim 17 were rejected under 35 U.S.C. 103(a) as being unpatentable over Fortin and based on an assertion that a function in a dynamic link library was common practice. Claim 17 adds to independent Claim 15 that the original function is in a dynamic link library. As discussed in Section D above, the procedure in Fortin is only for monitoring stripped objects, which means objects without any linkable information. Even if the use of a dynamic link library

is known, Fortin expressly excludes the use of linkable information and, thus, teaches away from it. Any attempt to combine Fortin and the use of a dynamic link library with merely an assertion is inappropriate and without merit. According, applicants respectfully submit that Claim 17 is not rendered obvious by Fortin and is allowable.

Claims 21-26

Independent Claim 21 were rejected under 35 U.S.C. 103(a) as being unpatentable over Fortin and further in view of Peek, USPN 5,481,706. As amended, Claim 21 includes

adding a wrapper of the imported function to an import data block;

adding a stub function for the imported function wherein the stub function comprises

an instruction that saves the address of the import function to a threaded local storage variable and replaces an access to the import function with an access to the user-supplied function; and

adding an entry in a function lookup table of the imported function.

As discussed above, Fortin fails to disclose modifying an original function with a user-supplied function. Fortin also fails to disclose a function lookup table, which is described in the applicants' specification as used for matching the name of the original function to the function's address. These deficiencies are not remedied by Peek.

The Office Action asserts that including a stub function is common practice and that Peek teaches adding a wrapper of a function to a data block. However, even if true, these assertions still do not describe the subject matter recited in Claim 21. Thus, applicants respectfully submit that independent Claim 21 is not rendered obvious by any combination of

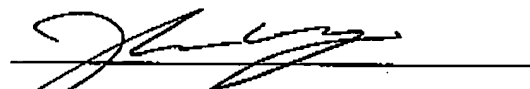
Fortin and Peek and is allowable. Dependent Claims 22-26 are also allowable for at least the same reason.

CONCLUSION

Applicants respectfully request that a timely Notice of Allowance be issued in this case.

Respectfully submitted,

MERCHANT & GOULD P.C.



Thomas Wong
Registration No. 48,577
Direct Dial: 206.342.6286

MERCHANT & GOULD P.C.
P. O. Box 2903
Minneapolis, Minnesota 55402-0903
206.342.6200



VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the abstract:

The abstract have been amended as follows:

[An original function or method in a machine-readable software component, such as function in an executable file, is instrumented regardless of whether the function is embedded within the machine-readable software component or an imported function residing in another DLL. When a call to an original function is instrumented, the entry point of the function and the identity of the function are stored in the machine-readable software component. A function lookup table is used for storing the identity and location of the original function. Storing the information enables a user-supplied function, that is called in substitute for the original function, to retrieve or lookup, the location of the original function, using the name of the original function as a key, and then call the original function. As a result, the user-supplied function acts as a wrapper of the original function and enables fault simulation code to control the execution of the original function. STUB for imported function.

Furthermore, a master lookup table that is created at initialization of an injector runtime library, identifies the location of the function lookup table using the base address of the instrumented executable file. This enables the function lookup table for each instrumented executable file to be located, and then searched to identify the location of an instrumented function with the instrumented executable file.]

This invention provides a system and method for instrumenting an executable file by replacing an original function in the executable file with a user-supplied function and enabling the user-supplied function to invoke the original function. In particular, the executable file includes an access to the original function. To instrument the executable file, the access to the original function is replaced with an access to the user-supplied function. Access information associated with the original function is retained, which enables the user-supplied function to locate the original function.

In the Claims:

Claims 1, 3, 7, 9, 15, 23, 28-31 and 36 have been amended as follows:

1. (Amended) A computerized method for creating an instrumented executable file, the method comprising:

[redirecting an original function in] modifying an executable file having an original function [to] with a user-supplied function; and

retaining access information of the original function, the access information enabling the user-supplied function to invoke the original function.

3. (Amended) The computerized method for creating an instrumented executable file as in claim 1, wherein the user-supplied function is in a dynamic [linked] link library.

7. (Amended) The computerized method for creating an instrumented executable file as in claim 1, wherein [the redirecting further comprises] modifying the executable file is performed using user-specified set points.

9. (Amended) The computerized method for creating an instrumented executable file as in claim 7, wherein modifying the executable file further comprises enabling the user-supplied function to invoke the original function in the executable file.

15. (Amended) A computerized method for executing an instrumented executable file [, the instrumented executable file having an original function redirected to a user-supplied function, and the user-supplied function having a jump to the original function, the method] comprising:

modifying the instrumented executable file having an original function with a user-supplied function, the user-supplied function having a jump to the original function;

saving the address of the original function in a threaded local storage variable;
and

invoking the user-supplied function using the address.

21. (Amended) A computerized method for instrumenting an imported function in an executable file for testing by callers of the imported function, the method comprising:

adding a wrapper of the imported function to an import data block;

adding a stub function for the imported function wherein the stub function comprises

[and] an instruction that saves the [original function] address of the import function to a threaded local storage variable and [an instruction that causes a jump to] replaces an access to the import function with an access to the user-supplied function; and

adding an entry in a function lookup table of the [original] imported function.

23. (Amended) A computerized method for instrumenting an embedded function in an executable file for testing by callers of the embedded function, the method comprising:

[redirecting] modifying an embedded function with a user-supplied function using [to the] a wrapper; and

adding an entry in a function lookup table of the address of the embedded function.

28. (Amended) A computerized system comprising:

means for [redirecting an original function in] modifying an executable file having an original function [to] with a user-supplied function; and

means for retaining access information of the original function, the access information enabling the user-supplied function to invoke the original function.

29. (Amended) A computerized system comprising:

an executable file having a call to an original function, the original function having an identity comprising a name and a parameter prototype; [and]

means for modifying the original function with a user-supplied function; and

means for [a] configuring the user-supplied function [that includes a call to the original function] to retrieve stored access information of the original function.

30. (Amended) A computerized system comprising:

an executable file having a jump to an original function, the original function having an identity comprising a name and a parameter prototype;

a first software component having a user-supplied function that includes a jump to the original function; and

a second software component for:

receiving the identity of the original function;

receiving the identity of the user-supplied function;

instrumenting [the function in] the executable file [using] by modifying the identity of the original function with the identity of the user-supplied function; and

storing the original function address in the executable file in association with the name of the original instrumented function.

31. (Amended) A computerized system comprising:

a first module of machine-readable code comprising:

a call to an [first instrumented] original function [call], the call being directed to a [first replacement] user-supplied function; and

a first data structure associating the identity of the [first instrumented] original function with the location of the [first instrumented] original function; and

a second module comprising the [first replacement] user-supplied function, [operatively coupled] linked to the first module [through] and a jump to the [first] original function.

36. (Amended) A computer-readable medium having computer-executable instructions to cause a computer to perform a method comprising:

[redirecting an original function in] an executable file having an original function

[to] with a user-supplied function; and

retaining access information of the original function, the access information enabling the user-supplied function to invoke the original function.